Constrained Software Distribution for Automotive Systems

Robert Höttger[1(✉)], Burkhard Igel[1(✉)], and Olaf Spinczyk[2(✉)]

[1] IDiAL Institute, Dortmund University of Applied Sciences and Arts, Dortmund, Germany
{robert.hoettger,igel}@fh-dortmund.de
[2] Computer Science Institute, Osnabrück University, Osnabrück, Germany
olaf.spinczyk@uos.de

Abstract. A variety of algorithms and technologies exist to cope with design space exploration for software distribution in terms of real-time, embedded, multiprocessor, and mixed-critical systems. The automotive domain not only combines those domains but even introduces further constraints and requirements due to several design decisions, standards, or evolved methodologies. In addition, solutions are predominantly proprietary, often lack in perspicuity, and sophisticated approaches towards the comprehensive concern of constraints are rather rare.

This paper presents typical constraints along with distributing automotive applications across the processing units of vehicles, outlines three software distribution methodologies based on the constraint programming paradigm, and evaluates those in comparison to related design space exploration approaches. Benchmarks upon hypothetical and industrial models show that the constraint-based approaches outperform other forms in many cases regarding quality and effectiveness. Additionally, the presented approach benefits from a holistic consideration of constraints such as processing unit affinities, safety level aggregations, communication costs as well as processing unit utilization optimization among others whilst being applicable to heterogeneous, networked, hierarchical, embedded, multi and many core architectures.

Keywords: AMALTHEA · AUTOSAR · APP4MC · Embedded real-time systems · Constraint programming

1 Introduction

Software distribution for embedded multi and many core systems gained significant importance in the recent years especially in the automotive domain due to the increasing demands of advanced driver assistance systems, autonomous driving, as well as architectural changes towards the centralization and consolidation of functional domains and Electronic Control Units (ECUs). Additionally, standardization (e.g. AUTOSAR[1], automotive SPICE[2]), collaboration across Tier

[1] Automotive Open System Architecture www.autosar.org, accessed 01.2019.
[2] Automotive SPICE http://www.automotivespice.com, accessed 12.2018.

R. Damaševičius and G. Vasiljevienė (Eds.): ICIST 2019, CCIS 1078, pp. 567–590, 2019.
https://doi.org/10.1007/978-3-030-30275-7_44

suppliers, Original Equipment Manufacturer (OEMs), and various tool vendors, and requirements from legacy applications necessitate sophisticated approaches when applying software distribution methodologies to the already highly constrained domain of heterogeneous, embedded, real-time, and mixed-critical environments. In order to reach reliability, safety, modularity, scalability, real time, or other goals, OEMs and suppliers introduced AUTOSAR in 2002 to overcome the tremendous amount of common challenges in the automotive industry and build up a basis to exchange, simulate, integrate, and even develop respective software. AUTOSAR undergoes regular releases since then, defining certain requirements and constraints such software has to address in order to be AUTOSAR compliant.

The basic concern of software distribution in the AUTOSAR context is the **partitioning** of runnables, i.e., atomic functions to tasks and the **mapping** of such tasks to processing units across micro controllers and ECUs. More precisely, given a set of runnables and a set of processing units, the goal is to find (a) a runnable to task assignment that is calculated by the partitioning process and forms the task set, and (b) a task to processing unit assignment denoted as mapping. This two phase approach yields a multitude of advantages such as distribution flexibility, level-based pairings or separations, as well as the consideration of various constraints described in Sect. 3. While this rather generic perceiving challenge has been studied for decades and is NP-complete [13], the holistic concern of the mandatory domain-specific constraints has been either omitted or only partially investigated. By making use of the open source AMALTHEA[3] model that is AUTOSAR compliant, this paper's work applies to a widely established superset of automotive constraints on the one hand, and can further cover industry driven requirements on the other hand. AMALTHEA features the typical combinatorial patterns to which constraint programming is preferably applicable. It comes with the APP4MC[4] platform and has, similar to AUTOSAR, regular maintenance and update releases. The APP4MC version used for this paper's investigations is 0.8.3 and implementations have been migrated to 0.9.1.

A promising and flexible paradigm applicable to partitioning and mapping is Constraint Programming (CP). CP allows natural problem modeling by making use of a huge variety of constraints that need to be satisfied for a valid solution. Typical features are logical, arithmetical, set, graph, or real-value expressions and coherencies among others. CP solver can be configured in various ways to investigate the solution space, optimize given objectives, and consequently solve the modeled Constraint Satisfaction Problem (CSP). Therefore, an algorithm is chosen to investigate the problem space of partitioning and mapping, e.g., incremental assignment combined with backtracking search or complete assignment combined with stochastic search. In order to remove invalid values from a variable's domain, propagation identifies inconsistent value combinations regarding the defined constraints and assigned values.

[3] AMALTHEA model http://eclip.se/eV, accessed 01.2019.
[4] Eclipse APP4MC, https://www.eclipse.org/app4mc/, accessed 01.2019.

Alternatively to using CP, local search approaches often follow a greedy-based structure and therefore may miss optimal values and valuable parts of the solution space. Furthermore, local search often has a dedicated model or application to work with and its applicability to different problems is very limited. In contrast to mathematical programming such as (Mixed) Integer Linear Programming (M)ILP, quadratic programming, or evolutionary (genetic) algorithms (GA), CP not only covers most of the mathematical operations of such, but also comes with powerful paradigms to further constrain combinatorial problem spaces and consequently increase exploration efficiency.

The **contribution** of this paper is the **formal outline of various constraints in the automotive industry** and their application to constraint programming used as a paradigm to **solve partitioning and mapping problems** of industry driven AMALTHEA models. Partitioning here concerns the distribution of runnables, which are atomic functions, to tasks, which can run both in parallel on the same processor, or concurrently across a multi core platform. The mapping problem defines the distribution of tasks to processing units which has to fulfill the amount of constraints outlined in the following sections and can be optimized towards various goals such as minimizing response times, balancing resource consumption, and others. The contribution includes considering a broad set of automotive constraints such as pairings, separations, affinities, timings (deadlines), sequences (precedence), ASIL- (Automotive Safety Integrity Level), partitioning-, and mapping-properties, balancing, hardware capacities, and communication costs when distributing software across tasks and processing units of vehicles. Presented (near) optimal solutions that consider this broad range of industry driven constraints has, to the best of the authors' knowledge, not been covered by related work. Along with the second contribution, the CP technique is compared with other design space exploration (DSE) approaches such as MILP, GA, and a heuristic whereas strengths and pitfalls of each are outlined along with hypothetical and industrial models. The comparison reveals new insights and assessments for applying DSE approaches to the highly constrained automotive software distribution problem. Results show that using the CP methodology significantly mitigates error prone and ineffective manual processes, partially outperforms other DSEs, and potentially eases software development and maintenance in the respective application field.

This paper is organized as follows. The subsequent section outlines related work as well as preliminaries this paper makes use of. Afterwards, Sect. 3 describes model entities required for the constraint descriptions in Sect. 4 and the optimization in Sect. 5. Finally, Sect. 6 provides benchmarks of the presented constraint solving approach, whereas Sect. 7 compares the results with some available related work. Finally, the conclusion in Sect. 8 summarizes this paper's contributions and results.

2 Related and Prior Work

Related work stretches across a huge variety of application domains when targeting DSE via heuristics, (M)ILP, Genetic Algorithms (GA), or CP. Since this

work focuses on the automotive domain, various requirements of embedded, real-time, mixed-critical, and highly interconnected systems define specifics such DSE approaches have to address. In fact, avionics, robotics, or logistic domains have certain similarities with this paper's automotive constraints. Furthermore, model-based programming techniques are used to utilize model checking, validation, and generation on the one hand and to specify data the DSE approaches are applied to on the other hand.

Typical optimization goals reach from execution time, energy consumption, resource utilization to reliability or solution quality as stated in a mapping survey in [30]. With the CP-based approach of this paper, multi-objective optimization is applied to different models and a variety of requirements and constraints is considered at the same time. Any typical optimization approaches can be configured whereas the remainder especially targets communication costs and resource utilization.

(M)ILP is one of the most used paradigms to cope with challenges such as partitioning and mapping, e.g., presented in [4]. MILP has though shown scalability issues for large-scale problems as stated in [29] and [17]. Laurent Perron also stated in [26] that the usage of CP is beyond MILP for optimizing applications in industrial operations research projects.

There is also a variety of papers stating that processing unit affinities are beneficial regarding application performance, fault tolerance, or security such as [7] or [20]. Such affinities are also considered within this paper via *arithmetical* constraints ensuring that a solution must contain given task to processing unit pairing.

Xiao et al. have shown in [33] that satisfying reliability goals and reducing resource consumption is challenging for precedence-constraints, mixed-critical, parallel, and embedded systems. However, the AMALTHEA model used in this work is based on AUTOSAR and highly differs from the presented reliability goal in [33] that is based on the *constant failure rate per time unit* combined with ensuring that tasks are mapped to processors that maximize a certain reliability value. In contrast, approaches presented in this paper ensure reliability via considering the various constraints such as affinities, pairings, separations, activations, safety levels, etc.

Thiruvady et al. studied the component deployment problem for vehicles in [32] that is similar to the partitioning and mapping problems of this work via CP. However, due to the consideration of only three major constraints (memory, colocation, and communication), their approach covers just a subset of this paper's constraints.

Oliveira et al. [10] provide one of the few publications that compares (M)ILP with CP for the JSSP (job shop scheduling problem). Their results show that CP outperforms MILP in many cases and that CP is assessed as being the prior choice to MILP in generic cases. The GA approach, however, has neither been applied to the JSSP problems in [10] nor compared with CP or ILP. Also, the according JSSP does not cover the specific constraints presented in this work.

Limtanyakul et al. apply CP to test scheduling for the automotive industry in [23]. Although the problem is different from this work's partitioning and mapping approaches, results have shown that the automotive domain comprises typical requirements and constraints that CP can fully utilize such that CP-based DSE can potentially be more effective and efficient.

Along with the FMTV benchmark that is used in Sect. 6, several research was presented in the recent years regarding solutions towards event chain latency calculation [14], contention analysis [4] under different communication paradigms [15,24], worst case execution/response time (WCET/WCRT) analyses [8], label mapping [6] and more. However, none of those publications covers the broad constraints described in this paper.

Hilbrich et al. present the most similar constraint programming approach towards safety, time, and mixed critical systems in [16]. The ASSIST Toolsuite is publicly available[5] and addresses typical concerns of the avionics industry. However, there are certain differences to the AMALTHEA model that is used for this paper's work. For instance, label (memory) accesses, label sizes, access rates, runnable sequencing, stimuli diversity, or event chains of AMALTHEA result in an increased amount of constraints as well as a deviation from respective propagation approaches and variable domains.

Krawczyk et al. [22] present mapping algorithms in order to map tasks to processing units via ILP and GA that have been extended by Cuadra et al. in [9] towards the incorporation of the simulated annealing paradigm. These implementations are taken as a reference to compare this work's results in Sect. 7. GA-based applications to automotive systems have been also investigated in [25]. Those results show, similar to generic multi-objective genetic algorithms in [11], that evolutionary algorithms scale well especially for large-scale problems.

Finally, some commercial tools exist from companies such as Inchron GmbH, Symtavision GmbH, Vector Informatik GmbH, and others that were not accessible to the authors that probably address constrained software distribution. Apart from the scope, efficiency, and quality assessment of such commercial products, it is expected that certain model transformations and integration activities consume additional efforts when using a multitude of tools during the development process.

Along with the investigation of related work, no publication could be found that considers both a broader set of automotive constraints and an analysis of different DSE methodologies for the partitioning and mapping problems.

3 System Model

This section defines model entities the subsequent sections make use of in order to scope the problem space when distributing

(a) a set of runnables $\mathcal{R} = \{r_1, ..., r_o\}, |\mathcal{R}| = o$ to tasks (**partitioning**) and

[5] ASSIST Toolsuite https://github.com/roberthilbrich/assist-public, accessed 10.2018.

(b) a set of tasks $\mathcal{T} = \{\tau_1, ..., \tau_m\}, |\mathcal{T}| = m$ to a set of processing units $\mathcal{PU} = \{pu_1, ..., pu_n\}$, $|\mathcal{PU}| = n$ (**mapping**)

while considering a vast amount of constraints. The partitioning result is denoted by $\mathcal{RA} = (ra_{i,j}) \in \{0,1\}^{o \times m}$, i.e., a distinctive runnable assignment, and the mapping result is denoted as $\mathcal{TA} = (ta_{j,k}) \in \{0,1\}^{m \times n}$, i.e., a distinctive task assignment. We assume that each runnable and each task must be assigned statically to exactly one target, i.e., task and processing unit respectively. After the partitioning, a task τ_j consists of an ordered sequence of p_j runnables $\tau_j = \{r_{j,1}, ...r_{j,p}\}$. Processing units $\mathcal{PU}$ can be obtained from higher abstraction levels such as micro controllers, cores, or ECUs, whereas communication costs are considered according to the modeled architecture properties such as labels, label sizes, label accesses, and access rates. Each processing unit pu_k is associated with a capacity $puc_k = f_k \cdot ipc_k$, i.e., the multiplication of the processing unit's frequency with its static instruction per cycle value. Consequently, puc_k is normalized towards 1 s. We assume fixed priority scheduling as well as distinct preemptive and cooperative task sets. Consequently, preemptive tasks can preempt any lower priority tasks and cooperative tasks can preempt lower priority cooperative tasks at runnable bounds only. Each runnable is associated with an activation T_i and a worst case execution time c_i that represents the instruction costs the runnable requires for execution. Runnables inherit their activation to the tasks they are assigned to: $\forall\ i$ with $ra_{i,j} = 1 : T_j = T_i$. Since c_i often varies and is described via Weibull distributions in AMALTHEA, upper bounds are chosen as the worst case execution times in this paper. Despite Weibull estimations, AMALTHEA provides different forms of instruction representations that are omitted in this paper. Such properties are especially useful for simulation frameworks such as [31] or others. The instruction cost IC_j value of a task τ_j is defined by the sum of its contained runnable instruction costs:

$$IC_j = \sum_{r_i : ra_{i,j}=1} c_i \tag{1}$$

A task's worst case execution time $C_{j,k}$ is derived from IC_j and depends on the processing unit's capacity puc_k the task is mapped to ($ta_{j,k} = 1$). For calculations in this paper, task's activation patterns T_j are considered via one second normalization as shown in the following Eq. 2.

$$C_{j,k} = \frac{IC_j \cdot \frac{10^{24}}{T_j}}{puc_k} \tag{2}$$

with T_j in pico seconds that has been chosen as an appropriate time scale but can also be changed to a different accuracy. 10^{24} is derived from 10^{12} for the 1 s normalization in pico seconds multiplied with another 10^{12} to compensate the denominator that is given in Hz, i.e., $\frac{1}{second}$. We assume that task's deadlines are implicit to their activation. Activations can be periodic, sporadic, event-driven, variable rate, relative periodic, and more[6] whereas the former two are considered

[6] See APP4MC documentation at http://eclip.se/fA, accessed 04.2019.

in this paper. For sporadic activation, the lower bound recurrence value (see Footnote 6) is used to consider worst case arrival rates of the corresponding tasks. Other activations go beyond the scope of this paper.

Data propagation between tasks is assumed to be achieved via the asynchronous use of shared labels. Labels correspond to parameters saved to memory. To determine the communication costs, a communication model like explicit, implicit, or logical execution time (LET) is preferred to be used in terms of AUTOSAR. The worst case response times (WCRT) of runnables and tasks depend on scheduling and corresponding preemptions. The response time analysis from Baruah et al. in [3] that has been extended by Balsini et al. in [2], can be used to calculate WCRT as well as event chain latency properties via recurrence relations. In addition, recent response time analysis solutions for adaptive variable rate tasks presented by Biondi et al. in [5] can be further incorporated to achieve more accurate WCRT analysis with a precise estimation of worst-case interference. However, the communication cost cc_k calculation presented in Sect. 5 uses $C_{j,k}$ only and can be configured to consider latencies, WCRT, and a corresponding communication paradigm. This configuration is omitted here in order to achieve comparable results in regard to the existing evolutionary algorithms, ILP solutions, and heuristics in Sect. 7. If configured differently, the solution space would be more constrained and results could be worse. Minor additions to this system model are made in the next sections along with specific concerns that are correspondingly described.

The above mentioned model entities, that are just a subset of AMALTHEA, excellently apply to the CP paradigm when addressing highly combinatorial problems. The huge amount of sets, relations, and properties constitute all data that a constraint model requires to potentially utilize CP benefits. For example, aggregations such as activations, tags, or ASIL properties can be directly converted to `.allEqual`$(x[])$ constraints that ensure that a valid solution must have equal values for all variables in x.

4 Constraint Modeling

This section describes a subset of constraints applied to the previously outlined model in order to calculate (a) the runnable to task partitioning and (b) the task to processing unit mapping. Constraints are modeled using the open source choco-library and its solver from Prud'homme et al. [27].

4.1 Runnable Partitioning Constraints

The first constraint shown in Eq. 3 outlines the activation aggregation (i.e. grouping) that applies consecutively to *ASIL properties, tags,* and *runnable pairings*. For ASIL level and Tag references, T_i of Eq. 3 is replaced with $Asil_i$ or Tag_i respectively, whereas $Asil_i$ denotes the ASIL level reference of runnable r_i and Tag_i its tag reference. Runnable pairings are modeled via $\forall i \in rp$:

`.allEqual` $(j|ra_{i,j} = 1)$. Equation 3 defines that a partition, i.e., task, must not contain runnables that commonly reference more than one activation.

$$\forall\ j \text{ with } ra_{i,j} \in \mathcal{RA} : \texttt{.allEqual}\,(\{T_i|ra_{i,j} = 1\}) \tag{3}$$

Or in other words, a task must not contain runnables of different activations. However, the amount of tasks per activation is not restricted by this constraint.

A combination of the Eq. 3 constraint and a dedicated existing heuristic from [19] that is available at the APP4MC platform is chosen in order to quickly partition models consisting of large runnable sets. In fact, a directed acyclic graph is built from runnables and their label accesses and graph branches are cut into tasks in order to ensure cause-effect-chains, i.e., read/write dependencies that represent causal relationships. The heuristic forms the parameter initialization of the constraint-based partitioning that incorporates the various constraints of this section.

As mentioned above, the same constraint of Eq. 3 is implemented for

- `ASIL` properties that classify automotive safety integrity levels into *A, B, C, D,* and *undefined* according to ISO 26262 safety requirements and corresponding identification of the software's relation to potential hazards and risks (A = lowest to D = highest)
- `runnable pairings` to pair e.g. functionally close runnables
- and `tags` in order to group, e.g., software components.

Since runnables inherit their properties to tasks, such constraints also hold for the mapping process. Consequently, the constraint of Eq. 3 is also applied to the task mapping for *activations, pairings,* and *ASIL* properties.

After ensuring the correct aggregations, Eq. 4 defines the constraint to ensure runnable sequencing. Runnable sequencing constraints can have multiple groups that can each reference an arbitrary amount of runnables. The constraint application is straight forward. For each subsequent group pairs $(rscg_x, rscg_y)$ across all runnable sequencing constraints, the following rule holds:

$$\forall\ r_a \in rscg_x, r_b \in rscg_y : a < b \text{ with } \tau_j = \{r_{j,1}, ..., r_{j,a}, ..., r_{j,b}, ...\} \tag{4}$$

The constraint of Eq. 4 is implemented as an arithmetical constraint using the smaller expression. Strictly defining runnable sequencing constraints ensures causal orders and significantly eases system determinism.

Finally, the actual runnable to task partitioning constraint can be defined. For this purpose, the $\mathcal{RA}$ boolean matrix is applied to a `.sum` constraint as shown in Eq. 5.

$$\forall\ r_i \in \mathcal{R} : \sum_{j \leq m} ra_{i,j} = 1 \tag{5}$$

I.e., a runnable is assigned to exactly one task.

In order to balance runnable loads to tasks, Eq. 6 sets the minimal task weight (i.e. execution time) to a lower bound value (lb_{τ_i}). Each scalar length equals the

number of runnables (scalar constraint) so that the dot product of both scalars has to be less or equal to the task's lower bound value:

$$\forall\ \tau_j \in \mathcal{T} : \sum_i \langle ra_{i,j}, c_i \rangle \leq lb_{\tau_j} \tag{6}$$

The lower bound definition in Eq. 6 has been found useful for larger models in order to decrease resolution time for load balancing.

Presented constraints are derived from the AUTOSAR standard [1]. They are mandatory for real and valid scenarios and have been evolved since 2004. New constraints can easily be added to the existing model in form of CP typical constraints or even new variables and corresponding modeling.

4.2 Task Mapping Constraints

When mapping tasks to processing units, several assumptions must be made. First of all, tasks must be mapped to exactly one processing unit, i.e., when having a boolean matrix $|\mathcal{T}| \times |\mathcal{PU}|$, i.e., $n \times m$, the sum of booleans (`true` $= 1$) across all processing units must be 1 for each task as stated in the following Eq. 7:

$$\forall\ \tau_j \in \mathcal{T} : \sum_k ta_{j,k} = 1 \tag{7}$$

Equation 7 for task mapping corresponds to Eq. 5 for the runnable partitioning.

Of course, additional constraints have to follow to consider the various model properties such as safety levels and avoid arbitrary results. Equation 8 begins with defining the processing unit capacity constraint and ensures that no processing unit is assigned with a set of tasks that would exceed the processing unit's execution capacity puc_k. Therefore, the task assignment scalar ta_k for all tasks at processing unit pu_k is multiplied with the execution time scalar for all tasks at the processing unit C_k (note here that $C_{i,j}$ is the execution time of τ_i at pu_j, and C_k is a scalar denoting all task execution times at pu_k). Hence, given $\langle ta_k, C_k \rangle = \sum_j (C_{j,k} \cdot ta_{j,k})$, and the task cost equation from Eq. 2 that contains the task's activation, the processing unit capacity constraint is defined by:

$$\forall\ pu_k \in \mathcal{PU} : \langle ta_k, C_k \rangle + cc_k(\mathcal{TA}) \leq puc_k \tag{8}$$

This capacity constraint incorporates inter processing unit, inter micro controller, and inter ECU communication costs denoted as cc_k. Additionally, timing constraints are added such that every task τ_j meets its deadline D_j via Eq. 9:

$$WCRT_j \leq D_j \tag{9}$$

with D_j denoting the deadline of task τ_j. In order to calculate $WCRT$, the classical recurrence relation method for preemptive tasks $WCRT_{j,k} = C_{j,k} + \sum_{h \in hp(j)} \left\lceil \frac{WCRT_{h,k}}{T_h} \right\rceil C_{h,k}$ is used. Memory access and contention costs go beyond the scope of this paper and are assumed to be accounted within $C_{j,k}$ values.

The communication costs cc_k for each processing unit depend on the mapping result $\mathcal{TA}$ as well as label to memory mapping. This specific label mapping is omitted in this paper due to lack of space. The cc_k values are defined by the label access rate derived from T_j, the label bit width bw_l, and the available hardware (e.g. crossbar) bit width bw_{hw}, i.e., $bm_l = \left\lceil \frac{bw_l}{bw_{hw}} \right\rceil$, and the label access cycles ac_l that is 9 cycles for accessing global memory or local memory of different processing units and 1 cycle for local pu_k memory:

$$cc_k = \sum_{j \,:ta_{j,k}=1} T_j \left(\sum_{l \text{ accessed by } \tau_j} bm_l \cdot ac_l \right) \text{ with } ac_l = \begin{cases} 1 & \text{if } l \text{ is at } pu_k's \text{ LRAM} \\ 9 & \text{otherwise} \end{cases} \tag{10}$$

Here, ac_l values are derived from the FMTV challenge description in [21]. More (formal) information about label accesses and dependency derivation can be found at [18]. By replacing 9 with $9 + n - 1$, further maximal FIFO arbitration at the crossbar can be considered as described in [28]. For calculating communication costs for a single task, the notation cc_j is used that uses the same calculation as in Eq. 10 but without the first sum over all tasks j.

After applying *tag* and *pairing* constraints to tasks as in Eq. 3, activation aggregations, i.e. grouping runnables referencing the same activation, can be further extended with a validation of a hyper period existence as shown in Eq. 11:

$$\forall\ \tau_j \text{ with } ta_{j,k} = 1 : T_j \in hyp \in P_{hyp} \tag{11}$$

Equation 11 states that each task τ_j of a task set mapped to a processing unit pu_k must reference an activation within a hyperperiod set $hyp \in P_{hyp}$. Each hyperperiod set hyp is defined by $hyp \subset T : \forall\ T_u \in hyp\ \exists\ v \cdot \text{lcm}_{hyp},\ v \in \mathbb{Z}$. A hyperperiod set is a set of periodic activations, that has a single integer least common multiple, i.e., the hyper period of the hyperperiod set. Hyperperiod sets do not necessarily have to be distinct, such that periodic activations may occur in several hyperperiod sets. Such validation can be used to lessen the pessimism of response time analysis approaches such as [3] and examine WCRT values for scheduling approaches regarding a given task to processing unit mapping set. This paper's investigations cover fixed-priority WCRT analysis for rate monotonic scheduling (RMS) as referred to in Eq. 9. Work on considering more sophisticated scheduling approaches, dynamic scheduling such as earliest deadline first (EDF), and event-chain latencies go beyond the scope of this paper.

Equation 12 defines the separation constraints in order to ensure that tasks are not mapped to the same processing unit. Each task separation constraint contains at least 1 group, i.e., a task subset, and optionally a target processing unit or target scheduler. $ts_{f,g}$ denotes the f-th task separation constraint and the g-th group within the separation constraint.

$$\begin{aligned} &TS = \{ts_0, ...\} : \quad ts = \{g_0, ..., tpu\} : g \subset \mathcal{T}, tpu \in [1, k] \\ &(a)\ \forall\ \tau_j \in ts_{f,0} : \quad \texttt{.allDifferent}\,(k : ta_{j,k} = 1) \\ &(b)\ \forall\ g, \tau_j \in ts_f : \quad \texttt{.notMember}\,(k : ta_{j,k} = 1, k_{og}) \\ &(c)\ \forall\ \tau_j \in ts_f : \quad \texttt{.arithm}\,(k : ta_{j,k} = 1, \text{"!="}, tpu) \end{aligned} \tag{12}$$

If no target and a single group is defined, case (a) shows that all tasks must be mapped to different processing units via using the `.allDifferent` constraint. If no target and multiple groups are defined, case (b) of Eq. 12 shows that each group must be mapped to different processing units via applying the `.notMember` constraint. Here, k_{og} is the set of processing unit indexes the other groups of the same separation constraint are mapped to. If a specific target is defined in a separation constraint, case (c) of Eq. 12 shows that all tasks across all groups must not be mapped to the specified processing unit via using the arithmetical constraint with an not equal operator.

Affinity constraints correspond to the task separation model whereas the constraints (a)–(c) of Eq. 12 are replaced by (a) `.allEqual`, (b) `.Member`, and (c) "=" instead of "!=".

Typical use cases for separation and affinity constraints are separations from interfering tasks or affiliations to specific hardware, e.g., floating point units, I/O interfaces, memory intensive tasks, or similar.

5 Optimization

Within this paper, the focus is on two optimization parameters: minimizing the maximal processing unit utilization across all processing units in percent ($\max_k(puu_k)$), i.e., load balancing, and minimizing the overall communication costs (occ). Both optimization criteria are important to maximize resource utilization and correspondingly the throughput, i.e., the amount of jobs finished in a given time frame. In fact, both bad load balancing and high communication costs lower the job throughput (amount of finished tasks to a given time) and result in higher hardware costs as well as aggravated determinism and timing analysis. Apart from the optimization, it is important to note that this work's DSE approaches are accompanied with the validation of previously described constraints. A multitude of additional optimization criteria can be added but are omitted here for comprehension reasons.

Beginning with the processing unit utilization optimization, Eq. 13 presents the respective calculation using the dot product along the C_k and ta_k scalars:

$$puu_k = \frac{\langle ta_k, C_k \rangle}{puc_k} \tag{13}$$

As described in Eq. 2 and Sect. 3, this calculation uses normalization towards one second and considers task activation rates. The processing unit utilization metric has been chosen over, e.g., remaining processing unit capacity because of its simplicity and its applicability to heterogeneous architectures.

There are multiple ways to minimize the overall processing unit utilization value. The most obvious one is defining a variable that is applied as a maximum constraint across all processing unit utilization values, i.e., $puu_{max} = \texttt{.max}(puu)$.

This value is then applied to the solver with the minimization objective.

Instead of minimizing the maximal processing unit utilization, maximizing the minimal processing unit utilization is also possible. However, the latter approach usually results in a larger solution space so that optimization time is increased significantly because the amount of variable combinations is much higher. Consequently, getting good or optimal results for large industrial models takes significantly longer when targeting to maximize puu_{min}. In other words, if the upper bound on processing unit utilization is found, the $min(puu_{max})$ approach would stop the resolution process whereas the $max(puu_{min})$ process would continue aligning (maximizing) the remaining utilization values. In all of the cases studied in this paper, minimizing the maximal processing unit utilization provided sufficient results since the load balancing across lower utilized processing units is not accounted within the measurements.

In addition to single parameter optimization, optimizing multiple parameters can also be addressed with CP. Therefore, a pareto front is calculated that optimizes multiple values towards the same objective, i.e., `min` or `max`. From a value set of a pareto front, optimization parameters can also be weighted in order to identify a single solution from a pareto front as being the final solution. For example, having a pareto front with three solutions and two optimization parameters to be minimized such as $op_0 = [2, 3, 8]; op_1 = [4, 3, 1]$, the third solution would be omitted when equally weighting op_0 and op_1 since its cumulated optimization parameter is higher than the results from solutions 1 and 2 ($2+4=6; 3+3=6; 8+1=9; \rightarrow opt_3 > (opt_1, opt_2)$). However, when weighting op_1 with 3, the results are $2+3\cdot 4=14, 3+3\cdot 3=12, 8+3\cdot 1=11$ and consequently solution three would be the best one. For the measurements presented in this work, optimization parameters from the pareto front are equally weighted as shown in Eq. 15.

Inter task communication (for the partitioning) and inter processing unit communication (for the mapping) costs are combined with load balancing so that both criteria are optimized towards their minimum value within the CPMO approach (constraint programming using multi objective optimization).

Communication costs are derived from Eq. 10 with the addition that they are 0 if a task pair is mapped to the same processing unit as stated in Eq. 14.

$$cc(\mathcal{TA}) = (td_{j,h})^{m\times m} \;:\; td_{j,h} \quad = \begin{cases} 0, & \text{if } (j=h) \vee (ta_{j,k} = ta_{h,k}) \\ cc_j, & \text{if } ta_{j,k} \neq ta_{h,k} \end{cases} \tag{14}$$

This approach can be advanced in order to consider hardware ports that connect arbitrary hardware instances using interfaces like CAN, Flexray, LIN, MOST, Ethernet, SPI, I2C, AXI, AHB, APB, SWR, or custom ones[7]. While this advancement is already in development, it is omitted here in order to compare results with the existing DFG, ILP, and GA approaches.

By using the `choco_solver` [27], one can make use of the powerful reification paradigm in order to set constraints for certain situations only. Without reification, calculating communication costs (i.e. $occ(\mathcal{TA})$) would be significantly more complex. The approach of calculating overall communication costs based on a

[7] http://eclip.se/f0 gives more information on hardware ports, accessed 01.2019.

Algorithm 1. OCC calculation

```
   Data: task communication costs td, task assignments ta
   Result: overall communication costs occ(TA)
1  forall the processing units pu ∈ PU do
2      forall the task combinations τ_j, τ_h ∈ td do
3          only consider cases where at least one task is mapped to the current
           processing unit pu:
4          if (ta_{j,k} ∨ ta_{h,k}) ⇒ τ_j or τ_h is mapped to pu_k then
5              if (ta_{j,k} ⊻ ta_{h,k}) then
6                  set task dependency of pu_k for tasks at different processing units
                   to the task dependency value cc_{j,h} = td_{j,h}
7              else
8                  set task dependency for tasks mapped to the same pu_k to 0:
                   cc_{j,h} = 0
9              end
10         end
11     end
12 end
13 occ(TA) = Σ_{j,h} cc_{j,h} = the sum of all task's communication costs
```

static dependency matrix td and reification (cf. Algorithm 1 line 3) is shown in Algorithm 1. It uses the pseudo code notation for a better understanding.

Line 8 ensures that the task dependency matrix is set to 0 for every task pair that is mapped to the same processing unit. Instead of using a simple `.ifThenElse` constraint at line 4, lines 4–8 are important to only keep a task dependency, i.e., setting $cc_{j,h}$ to $td_{j,h}$, if and only if a task pair is mapped to different processing units in line 6. If tasks are mapped to the same processing unit, $cc_{j,h}$ is set to 0 in line 8. Such situation is implemented using the `exclusive_or` statement in line 5 that is used as a reification for the arithmetic constraint setting cc, i.e., the communication cost matrix, to the task dependency value in line 8. Consequently, if no task of a task pair is mapped to a processing unit (line 4 is *false*), $cc_{j,h}$ is not changed in any way since it may still be either 0 or $td_{j,h}$.

As mentioned in Sect. 4, the method of calculating overall communication costs shown in Algorithm 1 is independent of the underlying communication paradigm such as explicit, implicit, or LET. Those paradigms are necessary when calculating latencies and contention effects along with WCRT that is not in scope of this paper.

Table 1 summarizes briefly the AMALTHEA constraints, requirements, or methodologies as well as the correspondingly implemented constraints. Some constraints were used in combination with additional variables or reification such as the `.ifThen` constraint combined with the `.addClausesXorEqVar` boolean variable (shown in Algorithm 1, line 5). Additionally, different approaches are implemented, e.g., for the partitioning, considering either integer or boolean variables resulting in different constraint types. This was done in order to

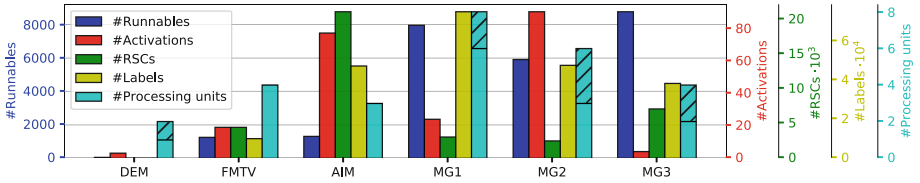

Fig. 1. Properties of DEM, FMTV, AIM, MG1, MG2, and MG3 models

overcome some scalability challenges and comparing the `.binpacking` constraint with a combination of arithmetical constraints. Measurements showed that the `.binpacking` constraint introduced overheads compared with the combination of arithmetical expressions and boolean variables.

Nevertheless, Table 1 also shows the benefits of using constraint definitions in contrast to, e.g., MILP, since the number of constraints remains relatively small and their usage is more natural compared with a combination of multiple inequality definitions.

6 Benchmarks

With the previously described constraint model, solution and optimization processes have been applied to three industrial models as well as three hypothetical models generated from a model generator as listed in Table 2.

Figure 1 presents the model properties of the respective models with the number of runnables, activations, runnable sequencing constraints ($RSCs$), labels, and processing units. The slash pattern above some processing unit bars indicate the amount of heterogeneous processing units that differ in frequency, instructions per cycle, or clock ratio from the basis processing units. In general, the

Table 1. Applied constraint types

AMALTHEA constraint	Choco [27] constraint
Runnable-, process-, ASIL-, or tag-pairing	`.allEqual`; `.arithm(=)`;
Runnable-, or process-separation	`.allDifferent`; `.arithm(≠)`; `.notMember`;
Runnable sequencing	`.arithm(<, =)`; `.allDifferentEx0`; `.max`;
Processing unit utilization	`.scalar`; `.count`; `.min` / `.max`;
Partitioning	`.binpacking`; `.min`; `.sum`; `.scalar`;
Activations	`.allEqual`; `.arithm(≤ )`; `.and`; `.or`;
Inter task communication	`.addClausesXorEqVar`; `.ifThen`;`.arithm(=)`; `.count`; `.and`;

Table 2. Examined models

DEM	Democar, an academic engine management system available at [12]
FMTV	An industry driven anonymized model available at [21]
AIM	An anonymized industrial model the authors have been granted access to
MG1, MG2, MG3	Generated models

number of processing units shown in Fig. 1 has been used to calculate quality values for the evaluation shown in Fig. 2, but more processing units can be modeled to calculate wider software mapping as exemplarily shown in Fig. 3.

Compared with other models, the Democar model contains fewest properties due to the fact that it represents a single engine control unit only and it has been manually modeled with academic content only in [12].

Each model is partitioned and mapped via six DSE approaches outlined in Table 3.

Additionally, the above outlined benchmarks have been extended to feature a varying number of (a) tasks and (b) processing units. Therefore, the measurements were performed upon several benchmark clusters. Results are presented and discussed in the following Sect. 7.

Results are saved within the model and utilized by compiler and linker scripts in order to be executed as binaries on a target hardware. After the compilation process, the static software distribution is not changed in accordance with AUTOSAR.

7 Evaluation

This section discusses obtained results along with quality, runtime, as well as scalability measurements. The partitioning process is not in scope of this evaluation but forms a requirement for the different task numbers along with measurements such as Fig. 4. Quality results are shown as scatter plots $puu_{max}(occ)$ in Fig. 2 as well as line charts that form speedup plots $su_{max}(\#\text{processingunits})$ in Fig. 3, and the $occ(\#\text{processing units})$ plot in Fig. 5(b). Runtimes are presented in line charts along Figs. 4 and 5(a). All line charts also provide information about the DSE's scalabiliy, due to their x axis representing either the number of tasks or the number of processing units as indicated.

7.1 Quality

Figure 2 presents the qualities of the six different DSE approaches measured as $puu_{max}(occ)$, i.e., the maximal processing unit utilization of the results depending on the number of overall communication costs. Each scatter plot (a)–(f) in

Table 3. DSE approaches applied to the models

DFG	Data flow graph heuristic [22]
ILP	Integer linear programming using oj!algo[8] [22]
GA	Genetic algorithm using jenetics[9] [22]
CP	Constraint programming without any optimization using the library from [27]
CPLB	CP + optimization for load balancing, i.e., minimizing the maximal processing unit utilization using the library from [27]
CPMO	CPLB + optimization for overall communication costs (occ) → multi objective optimization using the library from [27]

[8] ojAlgo library http://ojalgo.org, accessed 10.2018
[9] Jenetics library http://jenetics.io, accessed 10.2018

Fig. 2 concerns a distinct model (each model's properties are shown in Fig. 1) and features measurements of 3 different task amount configurations for each solution. The respective model and task configurations are indicated along with each subplot's title. For comprehension purposes, the concrete results are in light gray color whereas the mean values across the three tasks configurations are added as colored symbols.

With (a), the Democar model, the optimal puu_{max} value was achieved for CPLB and GA whereas CPLB features a lower occ value. DFG and ILP results are close to the optimum and GA has slightly higher occ values for 10 and 20 tasks. Interestingly, CPMO found solutions with significantly less communication costs across all task configurations. Whilst weighting the optimization objectives equally, this difference in communication costs even compensates the worse puu_{max} values > 0.9. This pareto front evaluation is shown in Eq. 15.

$$\forall s_q \in pf \ : 0 \geq \frac{puu_{s_f} - puu_{s_q}}{puu_{s_f}} + \frac{occ_{s_f} - occ_{s_q}}{occ_{s_f}} \tag{15}$$

Here, pf denotes the pareto front, s_q is any solution from the pareto front, and s_f is the final (chosen) solution. Consequently, for the two example solutions $s_1 : puu_{s_1} = 0.7;\ occ_{s_1} = 15$ and $s_f : puu_{s_f} = 0.8;\ occ_{s_f} = 5$, Eq. 15 ensures that $0 \geq \frac{0.8-0.7}{0.7} + \frac{5-15}{5} \Rightarrow \frac{1}{7} - 2 = -\frac{13}{7} \leq 0$ holds for all solutions of the pareto front.

The CP approach only identifies valid solutions, has no optimization at all, and consequently creates the worst solution quality across almost all models and configurations. However, CP features the lowest execution time compared with any other approach, even with DFG. Given the fact that all constraints can be easily covered with CP in contrast to DFG, CP may still be an appropriate choice for quickly identifying valid solutions whilst considering a variety of necessary constraints that all need to be fulfilled in a given solution.

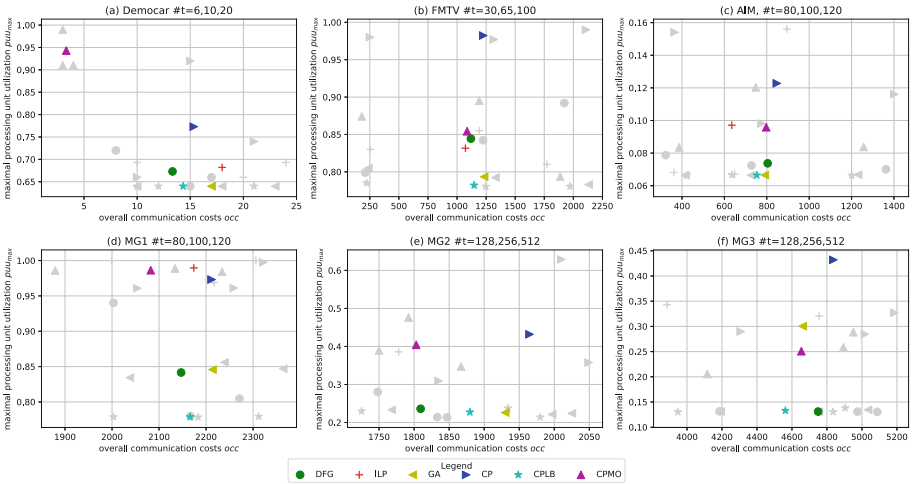

Fig. 2. Quality of six DSE approaches (DFG, ILP, GA, CP, CPLB, CPMO) along with the six models DEM, FMTV, AIM, MG1, MG2, and MG3

As soon as larger models are addressed, i.e., subplots (b)–(f) in Fig. 2, CPLB creates the best (lower puu_{max}, lower *occ* values) results for most measurements. Even setting the runtime to a single minute created better results compared with other DSE approaches. For instance, at the generated model (d) MG1, CPLB has lowest puu_{max} across all results. Here, the runtime was set to one minute whereas the ILP solver did not create any feasible results. The shown ILP values feature a runtime of 5 min and still do not reach comparable results to CPLB. For (e) MG2 and (f) MG3, ILP was not able to provide any results for 256 and 512 tasks, such that no ILP mean values are shown in the diagrams. This is due to the fact that the ILP solver does not scale well with the number of processing units and tasks as shown in the next Subsect. 7.2 and it does not reliably address heterogeneous processing units at all.

Interestingly, the DFG approach quickly found the optimal solution for 256 and 512 tasks in Fig. 2(e) i.e., the MG2 model. This is due to the fact that there is a relatively large task that contains a long sequence of runnables, i.e., a task that can not be further subdivided, and the DFG simply sorts tasks by their instruction costs and assigns those beginning with the largest chronologically to an ordered list of processing units beginning with the fastest (most instructions per second). However this greedy heuristic rarely results in good solutions as the other subplots, e.g., (b), (c), or (d) show.

Figure 3 presents the **speedup** of (a), the MG1 model, and (b), the FMTV model, along with an increasing number of homogeneous **processing units** (speedup (#processing units)). The used speedup calculation is based on [13] and provided in Eq. 16. Measurements from here on were repeated five times and shown values are mean values across those five benchmarks in order to mitigate measurement jitter caused by the operating system.

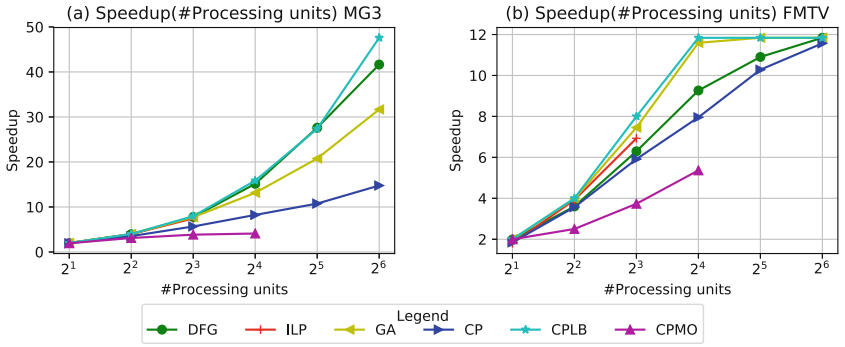

Fig. 3. Speedup (#processing units) of different DSE results for (a) MG3 and (b) FMTV

$$\text{speedup} = \frac{\sum_j \left(\min_k (C_{j,k}) \right)}{\max_k \left(\sum_j C_{j,k} |\ ta_{j,k} = 1 \right)} \tag{16}$$

Here, the nominator defines the minimal **sequential runtime** of all tasks ($\sum_j$) being mapped to the fastest processing unit. The denominator depends on the task mapping ta and identifies the maximal runtime across all processing units, i.e., **parallel runtime**. This speedup calculation is applicable to a heterogeneous processing unit structure and can be seen as the fraction of the time before the parallelization and the time after the parallelization as introduced in [13].

Due to limited dependencies between the tasks and runnables as well as a relatively homogeneous instruction distribution, almost optimal speedup factors can be reached whereas CPLB found the best values correspondingly to Figs. 2(f) and 3. Surprisingly, the DFG approach creates better results for the MG3 model compared to GA. However, this is not the case for the FMTV model as shown in Fig. 3(b) due to its more heterogeneous nature. While ILP was not able to scale beyond 8 processing units, CPLB required significantly more runtime after the amount of 16 processing units. The optimal, model independent, speedup value equals the number of processing units, but is barely achievable due to communication costs and varying task sizes.

With the FMTV model (cf. Fig. 3(b)), the speedup is saturated at 16 processing units due to the fact that there is a single task that can not be subdivided further and consequently forms the lower bound on schedule length. In order to avoid saturation, the model would have to provide less dependencies and more homogeneous tasks regarding their sum of instruction costs. While the difference between GA and CPLB is smaller in (b) compared with (a), CPLB still provides the best results along all number of processing units. As mentioned before, the DFG approach produces worse results in (b) whereas CP was able to achieve

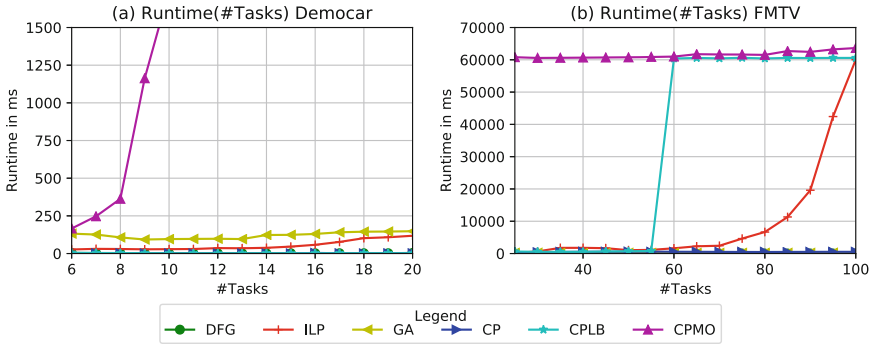

Fig. 4. Runtime(#tasks) of different DSEs for (a) Democar and (b) FMTV

better results than in (a). Other than that, results are similar to the MG3 model in (a).

7.2 Runtime and Scalability

Figure 4 presents the **runtime**, i.e., the efficiency, of different DSE approaches along with an increasing number of **tasks** (*rt(#Tasks)*) for (a) the Democar and (b) the FMTV model. Measurements were taken with an Intel i7 quad core computer running at 2,2 GHz with 16 GB Ram.

Results show that the multi-objective constraint programming approach (CPMO) scales worst with the number of tasks as it is the only approach with multi-objective optimization. The way the CPLB approach scales highly depends on the model structure. As soon as there is a single task defining the lower bound on the maximal processing unit utilization, i.e., a comparably large task that contains a high instruction cost and is consequently mapped to the fastest processing unit, CPLB will run quicker than ILP, DFG, or GA. If tasks' instruction costs are balanced, CPLB scales worse than ILP with the number of tasks, but better with the number of processing units. It is important to note though, that CPLB was always able to find at least a valid solution, while the ILP solver failed, e.g., regarding MG2 and MG3 (cf. Fig. 2(e), (f)), even with 12 h of resolution time. Furthermore, even if the constraint solver did not investigate the complete solution space, solutions feature better results, i.e., lower inter task communication and lower processing unit utilization, compared with DFG, ILP, or GA results (cf. CPLB marks at Fig. 2(a)). Concerning larger models, the single objective constraint approach (CPLB) outperforms almost every DFG, ILP, and GA result (except Fig. 2(e) and (f) for $|\mathcal{T}| = 512$) whereas the CPMO tends to create worse results beyond a task number of 65.

Figure 4(a) also shows that all approaches except CPMO stay almost linear below 200 ms in runtime whereas the CP, DFG, and CPLB approaches are yet the quickest with insignificant deviations. When applying the various DSEs to bigger models, the situation is similar: CPLB meets its limits to investigate the

complete solution space at about 60 tasks, whereas ILP does the same at around 100 tasks. CPLB however finds valid solutions already at the same time the CP approach does (< 5ms), while ILP may not provide solutions before its resolution time at all.

Figure 5 presents the **runtime** (a) and the *occ* values (b) of different DSE approaches along with an increasing number of **processing units** (*rt(#processing units)*) for the FMTV model.

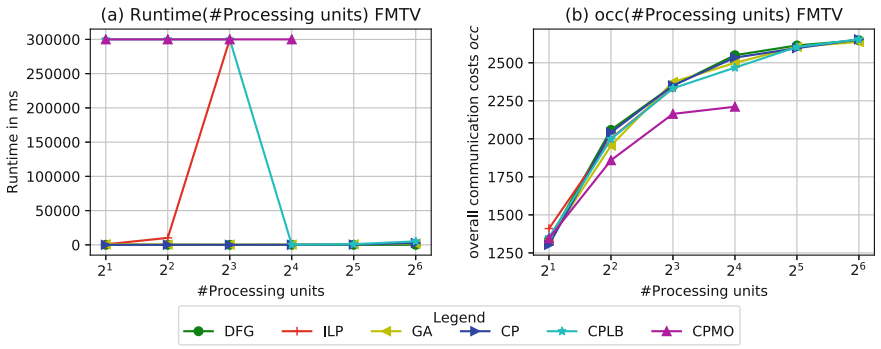

Fig. 5. (a) Runtime (#processing units); (b) occ (#processing units) of different DSEs for FMTV

Once again, CPMO performs with its maximal runtime definition (here set to 15 min) and CP finds valid solutions most quickly. The GA approach performs well but still takes longer than CP for each result. The worst scaling behavior shows the ILP approach. Above 8 processing units, the ILP solver did not find a valid solution at all. CPMO does also not scale well with the number of processing units and fails beyond 16 processing units for the same execution time restriction. Interestingly, the CPLB approach starts with requiring the full defined runtime but drops to the minimal runtime with 16 processing units and above. This is due to the fact that below 16 processing units, the solution space covers a huge variety of task to processing unit mapping combinations resulting in different maximal processing unit utilizations. As soon as 16 or more processing units are available, one relatively huge task defines the upper bound of processing unit utilization and mapping the other tasks to other processing units will not reduce this maximal processing unit utilization. Consequently, since the optimization targets only at minimizing the upper processing unit utilization bound but not maximizing the lower bound (this would in contrast keep the CPLB resolution time high), its optimization is done and solutions are available quickly. Figure 5(b) also shows a linear increase of communication costs with the increasing number of processing units as well as the CPMO approach with lowest *occ* values.

As soon as a heterogeneous structure of processing units is present, the puu_k metric shows its benefits since no additional calculations must be performed and

the utilization values consider processing unit specific properties. Additionally, the CP solver can be configured to a specific initialization in order to overcome the arbitrary initial assignment values that often creates an undesired homogeneous mapping along with the heterogeneous system. For example, instead of the processing unit capacity constraint only, the initial assignment could feature another lower bound comparable to Eq. 6, i.e., $\forall\, pu_k : \sum_j ta_{j,k} \geq 1 \;\; \text{with } m \geq n$. Assuming that the task number is higher than the number of processing units, this equation ensures that at least one task is mapped to each processing unit. This could also be extended in order to map the largest tasks to the fastest processing units following the DFG strategy. Therefore, tasks and processing units are arranged in a descending order (regarding IC_j and puc_k) and the ta matrix is initialized with corresponding values such that the largest task is mapped to the fastest processing unit and following tasks are mapped respectively to mitigate initial mapping efforts and reduce resolution time.

With applying the 6 DSE approaches to more generated models, results did not show significant deviation from results presented above. It was observed that the threshold for speedup limitation and the CPLB runtime always depends on whether there exists a task that defines the lower bound on execution time when increasing the number of processing units (cf. Figs. 3(b) and 5(a)).

8 Conclusion

The proposed CP-based DSEs provide a wide flexibility for engineers facing the highly constrained problem of distributing real-time and mixed-critical software to heterogeneous hardware with varying architectural structures and patterns. The lightweight CP approach without any optimization has shown to provide valid solutions faster than any other comparable approach such as DFG, ILP, or GA. The single objective optimization approach CPLB provides optimal or nearly optimal solutions for most of the measurements. In only 11% of all measurements, CPLB did not provide the best processing unit utilization values. For 73% of all measurements, CPLB defines the best overall result quality regarding occ and puu_{max} values. The CPMO approach covers multi objective optimization with assessable pareto fronts in an appropriate amount of time. For optimal results however, the multi object constraint programming solution requires significantly more time.

A great benefit of using the CP paradigm is also an automatic constraint validation that will inform programmers about any contradicting or erroneous model entities, variable bounds, or constraints. Such validation requires additional efforts when using different DSEs.

Additionally, this paper's work has shown that CP applies very well to highly constrained domains consisting of combinatorial design spaces such as automotive systems. It preserves the natural modeling and programming activities while providing optimal, pareto optimal, or nearly optimal solutions in an appropriate amount of resolution time. Typical automotive constraints, consecutive constraint modeling, and solving partitioning and mapping problems with a constraint solver are presented. The CP solver's perception is very natural since its

constraints often directly correspond to AMALTHEA entities and the CP SAT solver uses mainly clauses and backtracking search rather than linear inequalities only as in ILP. This also takes effect when adapting optimization goals which CP tackles on a more natural level. In addition to the efficiency and quality assessments of the described approaches, their benefits of effort mitigation when investigating highly constrained solution spaces are presented. We can conclude that if optimization is less important rather than getting valid solutions as quick as possible, CP is the prior choice. If constraints are of non integer nature or if the model is subject to a high amount of interleaving constraints of different types, CP, CPMO, and CPLB are the prior choice. If implementation simplicity is in focus, local search heuristics (such as DFG) can be useful for simple problems without any optimality demands. Genetic algorithms scale well for single objective optimization if the problem can be represented via few and simple mutation characteristics. Further work is intended to advance the utilization and response time analysis for various scheduling methods and to adjust optimization goals to consider further parameters such as memory contention or task chain latency.

To the best of the authors' knowledge, this paper is the first approach that considers AUTOSAR compliant constraints on a broader level and compares different DSE methodologies for the software distribution problem in the automotive domain.

References

1. AUTOSAR Consortium. Automotive Open System Architecture - Classic Platform 4.4.0 : Requirements on Timing Extensions (2019). https://bit.ly/32gVClq. Accessed 7 2019
2. Balsini, A., Melani, A., Buonocunto, P., Di Natale, M.: FMTV 2016 : where is the actual challenge? In: International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) (2016)
3. Baruah, S.K., Burns, A., Davis, R.I.: Response-time analysis for mixed criticality systems. In: Proceedings of the IEEE 32nd Real-Time Systems Symposium, RTSS 2011, pp. 34–43. IEEE Computer Society (2011)
4. Becker, M., Dasari, D., Nicolic, B., Åkesson, B., Nélis, V., Nolte, T.: Contention-free execution of automotive applications on a clustered many-core platform. In: 28th Euromicro Conference on Real-Time Systems, July 2016
5. Biondi, A., Di Natale, M., Buttazzo, G.: Response-time analysis of engine control applications under fixed-priority scheduling. IEEE Trans. Comput. **67**(5), 687–703 (2018)
6. Biondi, A., Pazzaglia, P., Balsini, A., Di Natale, M.: Logical execution time implementation and memory optimization issues in autosar applications for multicores. In: International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS) (2017)
7. Bonifaci, V., Brandenburg, B., D'Angelo, G., Marchetti-Spaccamela, A.: Multiprocessor real-time scheduling with hierarchical processor affinities. In: 28th Euromicro Conference on Real-Time Systems, ECRTS 2016, Toulouse, France, 5–8 July 2016, pp. 237–247 (2016)

8. Choi, J., Kang, D., Ha, S.: A novel analytical technique for timing analysis of FMTV 2016 verification challenge benchmark. In: International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) (2016)
9. Cuadra, P., Krawczyk, L., Höttger, R., Heisig, P., Wolff, C.: Automated scheduling for tightly-coupled embedded multi-core systems using hybrid genetic algorithms. In: Damaševičius, R., Mikašytė, V. (eds.) ICIST 2017. CCIS, vol. 756, pp. 362–373. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67642-5_30
10. Melo e Silva de Oliveira, R., Oliveira de Castro Ribeiro, M.S.F.: Comparing mixed & integer programming vs. constraint programming by solving job-shop scheduling problems. Indep. J. Manag. Prod. **6**(1), 211–238 (2015)
11. Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 416–423, July 1993
12. Frey, P.: A timing model for real-time control-systems and its application on simulation and monitoring of AUTOSAR systems dissertation. Ph.D. Thesis, Ulm University (2010)
13. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1990)
14. Hamann, A., Dasari, D., Kramer, S., Pressler, M., Wurst, F.: Communication centric design in complex automotive embedded systems. In: 29th Euromicro Conference on Real-Time Systems (ECRTS 2017), volume 76 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 10–20 (2017)
15. Hannig, F., Cardoso, J.M.P., Pionteck, T., Fey, D., Schröder-Preikschat, W., Teich, J. (eds.): ARCS 2016. LNCS, vol. 9637. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30695-7
16. Hilbrich, R., Behrisch, M.: Improving the efficiency of dislocality constraints for an automated software deployment in safety-critical systems. In: Combined Proceedings of the Workshops of the German Software Engineering Conference 2018 (SE 2018), Workshop on Software Engineering for Applied Embedded Real-Time Systems, SEERTS 2018, pp. 90–95. ceur-ws.org, March 2018
17. Hooker, J.N.: A hybrid method for the planning and scheduling. Constraints J. **10**(4), 385–401 (2005)
18. Höttger, R., Igel, B., Spinczyk, O.: On reducing busy waiting in AUTOSAR via task-release-delta-based runnable reordering. In: Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, pp. 1510–1515. IEEE, March 2017
19. Höttger, R., Krawczyk, L., Igel, B.: Model-based automotive partitioning and mapping for embedded multicore systems. In: International Conference on Parallel, Distributed Systems and Software Engineering, volume 2 of ICPDSSE 2015, pp. 2643–2649 (2015)
20. Jang, H.C., Jin, H.W.: MiAMI: multi-core aware processor affinity for TCP/IP over multiple network interfaces, pp. 73–82. In: Proceedings - Symposium on the High Performance Interconnects, Hot Interconnects (2009)
21. Kramer, S., Ziegenbein, D., Hamann, A.: Real world automotive benchmarks for free. In: 6th International Workshop an Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) (2015)
22. Krawczyk, L., Wolff, C., Fruhner, D.: Automated distribution of software to multicore hardware in model based embedded systems development. In: Dregvaite, G., Damasevicius, R. (eds.) ICIST 2015. CCIS, vol. 538, pp. 320–329. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24770-0_28

23. Limtanyakul, K.: Scheduling of tests on vehicle prototypes using constraint and integer programming. In: Kalcsics, J., Nickel, S. (eds.) Operations Research, pp. 421–426. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77903-2_65
24. Martinez, J., Sa, I., Burgio, P., Bertogna, M.: End-To-end latency characterization of implicit and LET communication models. In: Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) (2017)
25. Papadopoulos, Y., Grante, C.: Evolving car designs using model-based automated safety analysis and optimisation techniques. J. Syst. Softw. **76**(1), 77–89 (2005)
26. Perron, L.: Operations research and constraint programming at google. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, p. 2. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23786-7_2
27. Prud'homme, C., Fages, J.-G., Lorca, X.: Choco solver documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. (2016). http://www.choco-solver.org. Accessed November 2018
28. Rivas, J.M., Javier Gutiérrez, J., Medina, J.L., Harbour, M.G.: Comparison of memory access strategies in multi-core platforms using MAST. In: International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) (2017)
29. Sadykov, R., Wolsey, L.A.: Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. INFORMS J. Comput. **18**, 209–217 (2006)
30. Singh, A.K., Dziurzanski, P., Mendis, H.R., Indrusiak, L.S.: A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems. ACM Comput. Surv. **50**(2), 24:1–24:40 (2017)
31. Stattelmann, S., Ottlik, S., Viehl, A., Bringmann, O., Rosenstiel, W.: Combining instruction set simulation and WCET analysis for embedded software performance estimation. In: 7th IEEE International Symposium on Industrial Embedded Systems (SIES), pp. 295–298 (2012)
32. Thiruvady, D.R., Moser, I., Aleti, A., Nazari, A.: Constraint programming and ant colony system for the component deployment problem. In: Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Australia, 10–12 June 2014, pp. 1937–1947 (2014)
33. Xie, G., Chen, Y., Liu, Y., Wei, Y., Li, R., Li, K.: Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems. IEEE Trans. Industr. Inf. **13**(4), 1629–1640 (2016)